Collections

(http://docs.oracle.com/javase/tutorial/collections/
index.html)

Objectives

- Collections Framework (package java.util):
  - List: ArrayList, Vector → Duplicates are agreed
  - Set: HashSet, TreeSet → Duplicates are not agreed
  - Map: HashMap, TreeMap

The Collections Framework

- The Java 2 platform includes a new *collections framework*.

- A *collection* is an object that represents a group of objects.

- The Collections Framework is a unified architecture for representing and manipulating collections.

- The collections framework as a whole is not threadsafe.

The Collections Framework…

- **Reduces programming effort** by providing useful data structures and algorithms so you don't have to write them yourself.
- **Increases performance** by providing high-performance implementations of useful data structures and algorithms.
- **Provides interoperability between unrelated APIs** by establishing a common language to pass collections back and forth.
- **Reduces the effort required to learn APIs** by eliminating the need to learn multiple ad hoc collection APIs.
- **Reduces the effort required to design and implement APIs** by eliminating the need to produce ad hoc collections APIs.
- **Fosters software reuse** by providing a standard interface for collections and algorithms to manipulate them.

Collection Interfaces

- java.lang.**Iterable**<T>
    - java.util.**Collection**<E>
        - java.util.**List**<E>
        - java.util.**Queue**<E>
            - java.util.**Deque**<E>
        - java.util.**Set**<E>
            - java.util.**SortedSet**<E>
                - java.util.**NavigableSet**<E>
- java.util.**Map**<K,V>
    - java.util.**SortedMap**<K,V>
        - java.util.**NavigableMap**<K,V>

Methods declared in these interfaces can work on a list containing elements which belong to arbitrary type. T: type, E: Element, K: Key, V: Value

Details of this will be introduced in the topic Generic

3 types of group:
List can contain duplicate elements
Set can contain distinct elements only
Map can contain pairs <key, value>. Key of element is data for fast searching

Queue, Deque contains methods of restricted list.

Common methods on group are: Add, Remove, Search, Clear,…

Method	Description	
add(Object x)	Adds x to this collection	Elements can be stored using some
addAll(Collection c)	Adds every element of c to this collection	ways such as an array, a tree, a hash table.
clear()	Removes every element from this collection	Sometimes, we want to traverse elements as a
contains(Object x)	Returns true if this collection contains x	list → We need a list of references → Iterator
containsAll(Collection c)	Returns true if this collection contains every element of c	
isEmpty()	Returns true if this collection contains no elements	
iterator()	Returns an Iterator over this collection (see below)	
remove(Object x)	Removes x from this collection	
removeAll(Collection c)	Removes every element in c from this collection	
retainAll(Collection c)	Removes from this collection every element that is not in c	
size()	Returns the number of elements in this collection	
toArray()	Returns an array containing the elements in this collection	

The Collection Framework…

Central Interfaces

Common Used Classes

o java.util.**Collection**<E>
  o java.util.**List**<E>
  o java.util.**Queue**<E>
    o java.util.**Deque**<E>
  o java.util.**Set**<E>
    o java.util.**SortedSet**<E>
      o java.util.**NavigableSet**<E>
  o java.util.**Map**<K,V>
    o java.util.**SortedMap**<K,V>
      o java.util.**NavigableMap**<K,V>

o java.util.**ArrayList**<E>
o java.util.**Vector**<E>

Store: Dynamic array
Use index to access an element.

o java.util.**HashSet**<E>
o java.util.**TreeSet**<E>

Store: Specific structure/tree
Use iterator to access elements

java.lang.Comparable interface

o java.util.**HashMap**<K,V>
o java.util.**TreeMap**<K,V>

keySet()
values()

Use iterator

A TreeSet will stored elements using ascending order. Natural ordering is applied to numbers and lexicographic (dictionary) ordering is applied to strings.
If you want a TreeSet containing your own objects, you must implement the method compareTo(Object), declared in the Comparable interface.

Lists

- A List keeps it elements in the <u>order</u> in which they were added.

- Each element of a List has an index, starting from 0.

- Common methods:
  - **void add(int index, Object x)**
  - **Object get(int index)**
  - **int indexOf(Object x)**
  - **Object remove(int index)**

Classes Implementing the interface List

- AbstractList

- ArrayList

- Vector (like ArrayList but it is synchronized)

- LinkedList: *linked lists can be used as a stack, queue, or double-ended queue (deque)*

List Implementing Classes

```java
ArrayList list= new ArrayList();
for (int i = 101; i <= 110; i++) {
        list.add(i);
}
for (int i = 0; i < list.size(); i++) {
        System.out.println(list.get(i));
}
//or using Iterator
/*
    Iterator iter = list.iterator();
    while (iter.hasNext()) {
        System.out.println(iter.next());
    }
*/
```

Using the Vector class

```java
import java.util.Vector;
class Point {
  int x,y;
  Point() { x=0; y=0; }
  Point(int xx, int yy){
      x=xx; y=yy;
  }
  public String toString() { return "[" + x + "," + y + "]";}
}
public class UseVector {
  public static void main(String[] args) {
      Vector v = new Vector();
      v.add(15);
      v.add("Hello");
      v.add(new Point());
      v.add(new Point(5,-7));
      System.out.println(v);
      v.remove(2);
      System.out.println(v);
      for (int i=0;i<v.size();i++) System.out.print(v.get(i) + ", ");
      System.out.println();
  }
}
```

java.util.**Vector**<E> (implements java.lang.Cloneable, java.util.List<E>, java.util.RandomAccess, java.io.Serializable)

The Vector class is obsolete from Java 1.6 but it is still introduced because it is a parameter in the constructor of the javax.swing.JTable class, a class will be introduced in GUI programming.

Output – Chapter08 (run)

```
run:
[15, Hello, [0,0], [5,-7]]
[15, Hello, [5,-7]]
15, Hello, [5,-7],
```

Sets

- Lists are based on an ordering of their members. Sets have no concept of order.

- A Set is just a cluster of references to objects.

- Sets may not contain duplicate elements.

- Sets use the equals() method, not the == operator, to check for duplication of elements.

```java
void addTwice(Set set) {
    set.clear();
    Point p1 = new Point(10, 20);
    Point p2 = new Point(10, 20);
    set.add(p1);
    set.add(p2);
    System.out.println(set.size());
}
```

will print out 1, not 2.

Sets…

- Set extends Collection but does not add any additional methods.
- The two most commonly used implementing classes are:
  - TreeSet
    - Guarantees that the sorted set will be in ascending element order.
    - log(n) time cost for the basic operations (add, remove and contains).
  - HashSet
    - Constant time performance for the basic operations (add, remove, contains and size).

TreeSet and Iterator

- Ordered Tree – Introduced in the subject Discrete Mathematics

- Set: Group of different elements

- TreeSet: Set + ordered tree, each element is called as node

- Iterator: An operation in which references of all node are grouped to make a linked list. Iterator is a way to access every node of a tree.

- Linked list: a group of elements, each element contains a reference to the next

TreeSet = Set + Tree

Random r = new Random();
TreeSet myset = new TreeSet();
for (int i = 0; i < 10; i++) {
    int number = r.nextInt(100);
    myset.add(number);
}
//using Iterator
Iterator iter = myset.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next());
}

The result may be:

7
27
36
41
43
46
49
57
75
83

Using the TreeSet class & Iterator

```java
import java.util.TreeSet;
import java.util.Iterator;
public class UseTreeSet {
    public static void main (String[] args){
        TreeSet t= new TreeSet();
        t.add(5); t.add(2); t.add(9);t.add(30); t.add(9);
        System.out.println(t);
        t.remove(9);
        System.out.println(t);
        Iterator it= t.iterator();
        while (it.hasNext())
            System.out.print(it.next() + ", ");
        System.out.println();
    }
}
```

Output – Chapter08 (run)

```
run:
[2, 5, 9, 30]
[2, 5, 30]
2, 5, 30,
```

A TreeSet will stored elements using ascending order. Natural ordering is applied to numbers and lexicographic (dictionary) ordering is applied to strings.
If you want a TreeSet containing your own objects, you must implement the method compareTo(Object), declared in the Comparable interface.

Hash Table

- In array, elements are stored in a contiguous memory blocks → Linear search is applied → slow, binary search is an improvement.

- Hash table: elements can be stored in a different memory blocks. The index of an element is determined by a function (hash function) → Add/Search operation is very fast (O(1)).

"Smith" — f → 5

The hash function f may be:
'S'*10000+'m'*1000+'i'*100+'t'*10+'h' % 50

49	
14	Brown
9	Hoa
5	Smith
0	Linel

HashSet = Set + Hash Table

```java
import java.util.HashSet;

public class demo05 {

    public static void main(String[] args) {
        HashSet list = new HashSet();
        list.add(3);
        list.add(5);
        list.add(20);
        list.add(3);
        list.add(9);
        list.add(17);
        System.out.println(list);
    }
}
```

```
run:
[17, 3, 20, 5, 9]
BUILD SUCCESSFUL (total time: 0 seconds)
```

HashSet or TreeSet?

- If you care about <u>iteration order</u>, use a Tree Set and pay the time penalty.

- If iteration order doesn't matter, use the higher-performance Hash Set.

How to TreeSet ordering elements?

- Tree Sets rely on all their elements implementing the interface java.lang.Comparable.

public int compareTo(Object x)

- Returns a positive number if the current object is "greater than" x, by whatever definition of "greater than" the class itself wants to use.

How to TreeSet ordering elements?

```java
public class Employee implements Comparable<Object>{
    String code;
    String name;
    int age;

     public Employee(String code, String name, int age) {
          this.code = code;
          this.name = name;
          this.age = age;
     }

    public String toString(){
        return code+ " - " + name + " - " +age;
    }

     @Override
     public int compareTo(Object t) {
         Employee emp=(Employee)t;
        if(this.age>emp.age)
            return 1;
        else if(this.age==emp.age)
            return 0;
        else
            return -1;
     }
}
```

How to TreeSet ordering elements?

```java
import java.util.Iterator;
import java.util.TreeSet;


public class demo04 {
    public static void main(String[] args) {
        TreeSet list=new TreeSet();
        list.add(new Employee("SE1700", "Nguyen Hoai Bao", 25));
        list.add(new Employee("SE1701", "Nguyen Hoai Thu", 29));
        list.add(new Employee("SE1700", "Nguyen Hoai Thanh", 23));
        list.add(new Employee("SE1700", "Nguyen Hoai Ha", 20));
        //In danh sach nhan vien tăng dần theo age
        Iterator itr = list.iterator();
        while (itr.hasNext()) {
            Object element = itr.next();
            System.out.println(element);
        }
    }
}
```

How to TreeSet ordering elements?

Maps

- Map doesn't implement the java.util.Collection interface.
- A Map combines *two* collections, called keys and values.
- The Map's job is to associate exactly one value with each key.
- A Map like a dictionary.
- Maps check for key uniqueness based on the equals() method, not the == operator.
- IDs, Item code, roll numbers are keys.
- The normal data type for keys is String.

K   **V**

key ——————→ value
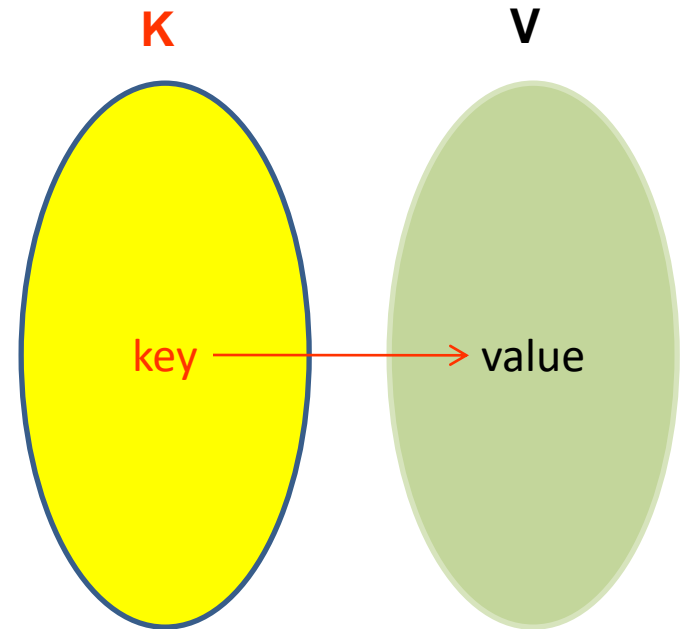
Each element: <key,value>

Maps..

- Java's two most important Map classes:

  - HashMap (mapping keys are unpredictable order – hash table is used, hash function is pre-defined in the Java Library).

  - TreeMap (mapping keys are natural order)-> all keys must implement Comparable (a tree is used to store elements).

HashMap

```
public static void main(String[] args) {
    HashMap mymap = new HashMap();
    mymap.put(1, "One");
    mymap.put(2, "Two");
    mymap.put(3, "Three");
    mymap.put(4, "Four");
    //using Iterator
    Iterator iter = mymap.keySet().iterator();
    while (iter.hasNext()) {
        Object key = iter.next();
        System.out.println(key + ": " + mymap.get(key));
    }
}
```

//output
1: One
2: Two
3: Three
4: Four

Key: integer, value: String

HashMap

```
demo06.java  ×
Source  History

 1
 2    import java.util.HashMap;
 3    import java.util.Iterator;
 4    import java.util.Scanner;
 5
 6
 7    public class demo06 {
 8        public static void main(String[] args) {
 9            HashMap mymap = new HashMap();
10            Scanner sc=new Scanner(System.in);
11            String NameAni;
12            mymap.put("Meo","Cat");
13            mymap.put("Cho","Dog");
14            mymap.put("Gau","Bear");
15            mymap.put("Su Tu","Lion");
16
17            System.out.print("Enter Name Animimal:");
18            NameAni=sc.nextLine();
19            //using Iterator
20            Iterator iter = mymap.keySet().iterator();
21            System.out.println("\n----------------------\n");
22            while (iter.hasNext()) {
23                Object key = iter.next();
24                if(key.equals(NameAni))
25                    System.out.println(NameAni + " => " + mymap.get(key));
26            }
27        }
28    }
```

HashMap


```
run:

Enter Name Animimal:Gau


-----------------------------


Gau => Bear
BUILD SUCCESSFUL (total time: 3 seconds)
```

Using HashMap class & Iterator

```java
import java.util.HashMap;
import java.util.Iterator;
public class UseHashMap {
    public static void main(String[] args){
        HashMap h = new HashMap();
        h.put("Sáu Tấn", "Huỳnh Anh Tuấn");
        h.put("Bình Gà", "Nguyễn Tấn Sầu");
        h.put("Ba Địa", " Trần Mai Hoà");
        System.out.println(h);
        h.put("Sáu Tấn","Nguyễn Văn Tuấn");
        System.out.println(h);
        h.remove("Bình Gà");
        System.out.println(h);
        Iterator it = h.keySet().iterator();
        while (it.hasNext())
        {  String key= (String)(it.next());
            String value = (String)(h.get(key));
            System.out.println(key + ", " + value);
        }
    }
}
```
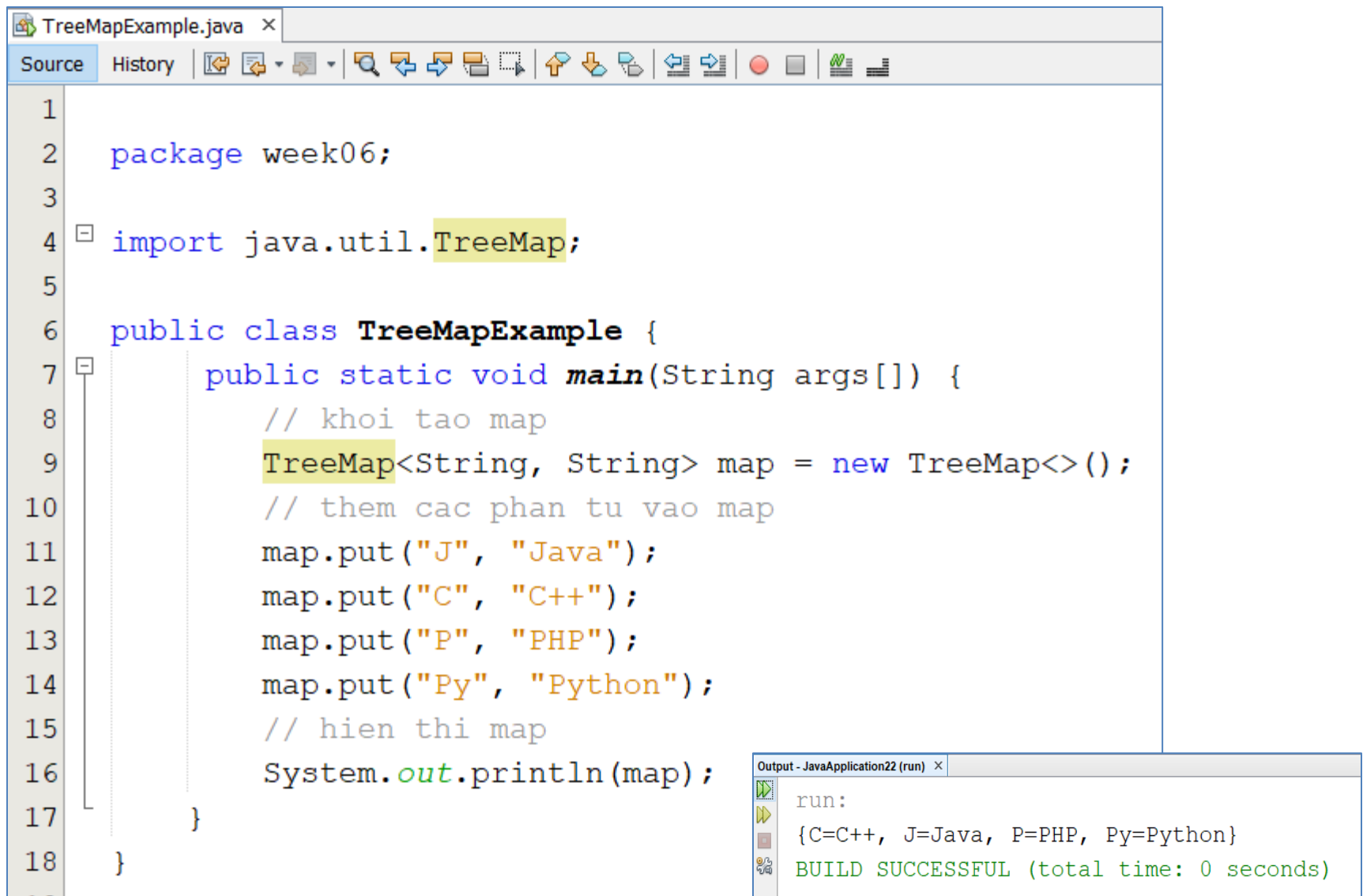
Key: String, value: String

Output - Chapter08 (run)

```
run:
{Ba Địa= Trần Mai Hoà, Sáu Tấn=Huỳnh Anh Tuấn, Bình Gà=Nguyễn Tấn Sầu}
{Ba Địa= Trần Mai Hoà, Sáu Tấn=Nguyễn Văn Tuấn, Bình Gà=Nguyễn Tấn Sầu}
{Ba Địa= Trần Mai Hoà, Sáu Tấn=Nguyễn Văn Tuấn}
Ba Địa,  Trần Mai Hoà
Sáu Tấn, Nguyễn Văn Tuấn
BUILD SUCCESSFUL (total time: 1 second)
```

Using TreeMap class

```java
package week06;

import java.util.TreeMap;

public class TreeMapExample {
    public static void main(String args[]) {
        // khoi tao map
        TreeMap<String, String> map = new TreeMap<>();
        // them cac phan tu vao map
        map.put("J", "Java");
        map.put("C", "C++");
        map.put("P", "PHP");
        map.put("Py", "Python");
        // hien thi map
        System.out.println(map);
    }
}
```

Output - JavaApplication22 (run) ×

```
run:
{C=C++, J=Java, P=PHP, Py=Python}
BUILD SUCCESSFUL (total time: 0 seconds)
```

Using TreeMap class & Iterator

```java
package week06;

import java.util.Iterator;
import java.util.TreeMap;

public class TreeMapExample02 {

    public static void main(String args[]) {
        // khoi tao map
        TreeMap<String, String> map = new TreeMap<>();
        String key;
        // them cac phan tu vao map
        map.put("J", "Java");
        map.put("C", "C++");
        map.put("P", "PHP");
        map.put("Py", "Python");
        // show TreeMap
        Iterator<String> itr = map.keySet().iterator();
        while (itr.hasNext()) {
            key=itr.next();
            System.out.println(key + "-" + map.get(key));
        }
    }
}
```

Using TreeMap class & Iterator

Summary

- The Collections Framework
  - **The *Collection* Super interface and Iteration**
  - **Lists**
  - **Sets**
  - **Maps**